MINI-PROJECT

ON

EMAIL - CLUSTERING

By

Y. Navya (N140104)

K. Roopa (N140127)

K.Yamini (N140116)

S.Alekhya (N140320)

Under the guidance of

Mr. K. Sravan Kumar

Faculty, Dept of Computer Science

RGUKT- IIIT, Nuzvid

A Project Work submitted to the Department of CSE for Mini project of E3,Sem1.

Rajiv Gandhi University of Knowledge Technologies, A.P. IIIT

Nuzvid-521201

Rajiv Gandhi University of Knowledge Technologies

APIIIT-Nuzvid

Department of CSE

CERTIFICATE OF PROJECT COMPLETION

This is to certify that the project work entitled “Email Clustering”is a bonafied work by N140116, N140127, N140104, and N140320 submitted for E3 sem1 mini project to the department of CSE under my guidance during the academic year 2017-2018. This project in my opinion, is worthy of consideration for the awarding of marks in accordance with the Department and University regulations.

Date:

Place:

R.Upendar Rao
Head of the Department
Department of Computer Science and Engineering

Supervisor

Mr. K.Sravan Kumar
Faculty
Department of CSE

september 2018

ABSTRACT:

Earlier there are classifications only for filtering messages based on priority,assigning messages to user created folders,or identifying spam messages.Here we mainly focuss on the problem of assigning e-mail messages to folders based on the user foldering strategy.Here,We used Enron E-mail dataset for clustering of e-mails.

Some people receive hundreds of e-mails in a day. Sorting of those e-mails is a time consuming process.The aim of this project is to develop a machine learning algorithm which would learn from the manual sorting of emails by the user in order to quickly be able to handle itself the processing of new emails.

We do not want to have to wait for the user to label a sufficiently high number of emails in each category before being able to make a prediction. We are therefore looking to implement an online learning algorithm, which will make a prediction for each incoming email as soon as it arrives (even for the first ones, even though the tentative labelling is likely to be erroneous). However, the algorithm will not display its prediction to the user and move the email to a specific folder unless it is confident in its prediction. Indeed, an application which frequently mislabels email or worse, ”loses” important emails by mistakenly placing them in a ”garbage” folder rarely consulted by the customer, is just not worth using. An interesting part of the project is therefore to evaluate the degree of confidence of the algorithm in its prediction.

For clustering the unlabeled emails We used unsupervised machine learning because We have training data with only inputs, also known as features and contains no outcomes. In supervised machine learning we work with inputs and their known outcomes. In this case We wanted to classify emails based on their message body, definitely an unsupervised machine learning task.

Here we use k-means clustering algorithm to divide e-mails into clusters.

We divided the keywords of particular mail into some clusters with no.of iterations.After discovering the most popular terms and the most

exciting emails due to clustering algorithms, We were looking for a manner to further group emails related to a specific keyword.

The first thing that came to mind to achieve this was cosine similarity. A common technique used to measure cohesion within clusters in the field of data mining.

To find the cosine distance of one email and all the others, We just need to compute the dot products of the first vector with all of the others as the vectors are already row-normalized. To get the first vector, We need to slice the matrix row-wise to get a submatrix with a single row.Using cosine similarity we will find the similarity between two e-mails.We are going to group the e-mails based on the particular key word.For this we used supervised learning.

EXISTING AND PROPOSAL SYSTEMS:

→EXISTING SYSTEM:

Earlier there are classifications only for filtering messages based on priority,assigning messages to user created folders,or identifying spam messages.

For that they used hierarchical clustering algorithms.

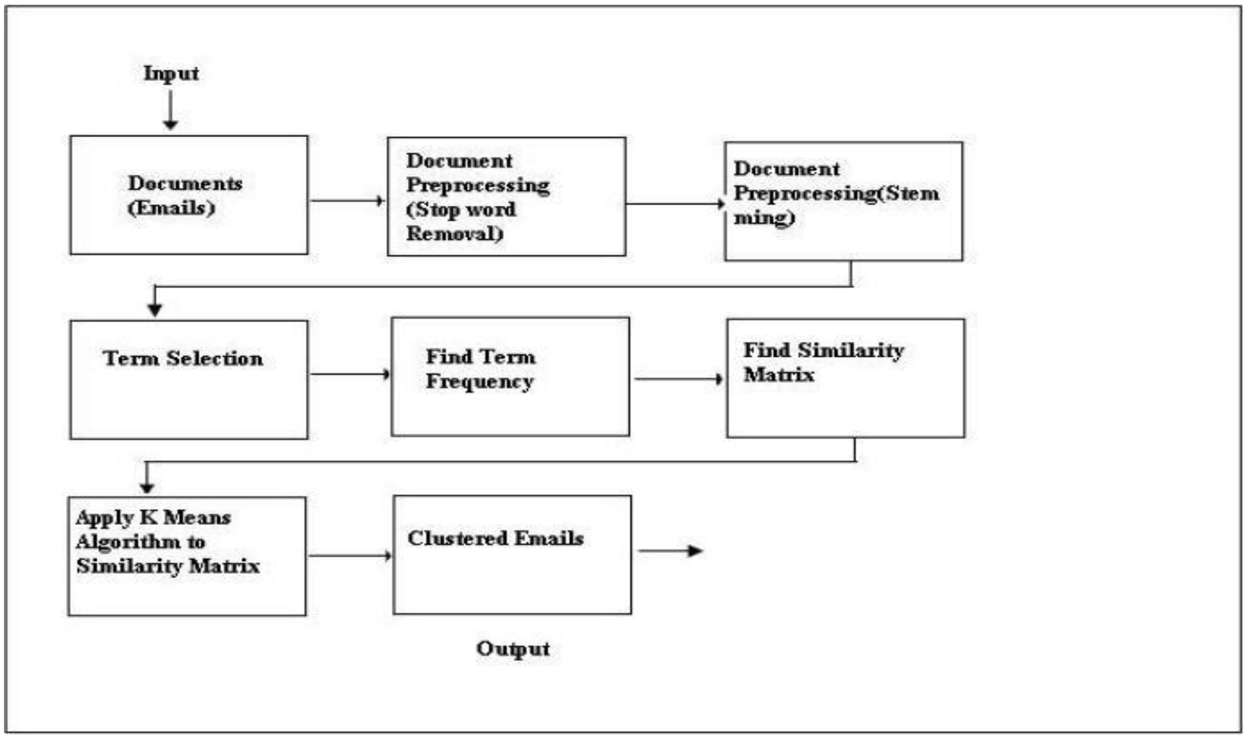
→DRAWBACKS OF EXISTING SYSTEM:

-It doesn’t produce effective results.

-Time complexity for these systems is $O(n^2)$.

-Hierarchical clustering is only good for small datasets.

→PROPOSING SYSTEM:


Input
Documents (Emails)
Document Preprocessing (Stop word Removal)
Document Preprocessing(Stem ming)
Term Selection
Find Term Frequency
Find Similarity Matrix
Apply K Means Algorithm to Similarity Matrix
Clustered Emails
Output

→ADVANTAGES OF PROPOSING SYSTEM:

-Easily implemented.

-K-means is linear in the number of data objects so the time complexity is O(n).

-With large number of variables K-means may be computationally faster than hierarchical clustering.

-K-means may produce tighter clusters than hierarchical clustering.

REQUIREMENTS:

Software requirements:

1.Anaconda software for python programming

2.Enron e-mail dataset.

Hardware requirements:

1. PC with minimum4GB RAM and 64 bit processor

MODULES :

ENRON DATASET:

We used Enron dataset for our clustering.

A large set of email messages, the Enron corpus, was made public during the legal investigation concerning the Enron corporation. The raw corpus is currently available on the web at http://www-2.cs.cmu.edu/~enron/. The current version contains 619,446 messages belonging to 158 users. We cleaned the corpus for use in these experiments by removing certain folders from each user, such as \discussion threads" and \notes inbox". These folders were present for most users, and did not appear to be used directly by the users, but rather were compute generated. Many, such as all documents", also contained large numbers of duplicate email messages, which were already present in the users' other folders. Since our goal in this paper is to explore how to classify messages as organized by a human, these folders would have likely been misleading.

In our cleaned Enron corpus, there are a total of 200,399 messages belonging to 158 users with an average of 757 messages per user. This is approximately one third the size of the original corpus. Figure 1 shows the distribution of emails per user. The users in the corpus are sorted by ascending number of messages along the x-axis. The number of messages is represented in log scale on the y-axis. The horizontal line represents the average number of messages per user (757).As can be seen from the graph, the messages are distributed basically exponentially,with a small number of users having a large number of messages.

However, there are users distributed along the entire graph from one message to 100,000 messages, which shows that the Enron dataset provides data for users with all amounts of email. More important in folder classification, though, is the number of folders each user has.

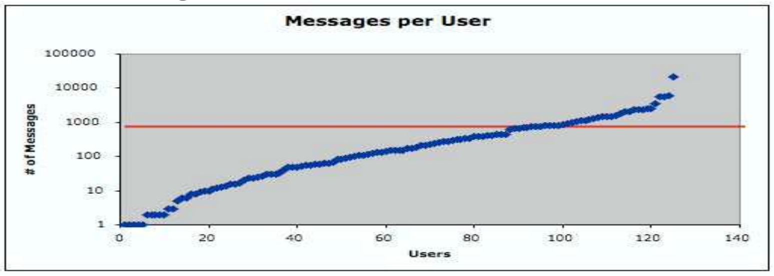


Fig1

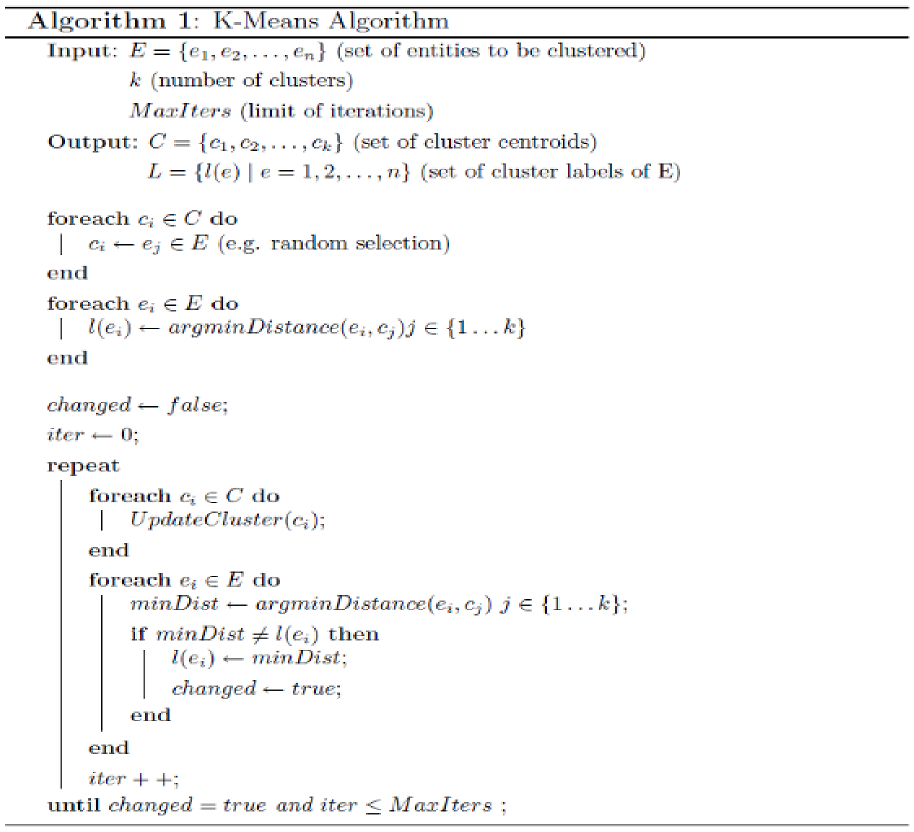
UNSUPERVISED LEARNING:

-We wanted to classify emails based on their message body, definitely we have to use an unsupervised machine learning task.

In unsupervised learning no teacher is available. The learner only discovers persistent patterns in the data consisting of a collection of perceptions. This is also called exploratory learning. Finding out malicious network attacks from a sequence of anomalous data packets is an example of unsupervised learning.

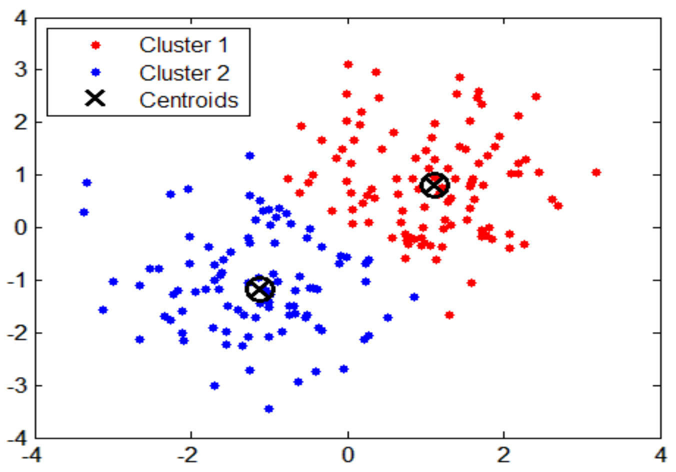
K-MEANS CLUSTERING:

-We use k-means clustering algorithm to divide e-mails into clusters.



Algorithm 1: K-Means Algorithm

```
Input: $E = \{e_1, e_2, \ldots, e_n\}$ (set of entities to be clustered)
       $k$ (number of clusters)
       $MaxIters$ (limit of iterations)
Output: $C = \{c_1, c_2, \ldots, c_k\}$ (set of cluster centroids)
        $L = \{l(e) \mid e = 1, 2, \ldots, n\}$ (set of cluster labels of E)

foreach $c_i \in C$ do
    $c_i \leftarrow e_j \in E$ (e.g. random selection)
end
foreach $e_i \in E$ do
    $l(e_i) \leftarrow argminDistance(e_i, c_j) j \in \{1 \ldots k\}$
end

$changed \leftarrow false$;
$iter \leftarrow 0$;
repeat
    foreach $c_i \in C$ do
        $UpdateCluster(c_i)$;
    end
    foreach $e_i \in E$ do
        $minDist \leftarrow argminDistance(e_i, c_j)\ j \in \{1 \ldots k\}$;
        if $minDist \neq l(e_i)$ then
            $l(e_i) \leftarrow minDist$;
            $changed \leftarrow true$;
        end
    end
    $iter++$;
until $changed = true$ and $iter \leq MaxIters$ ;
```



COSINE SIMILARITY:

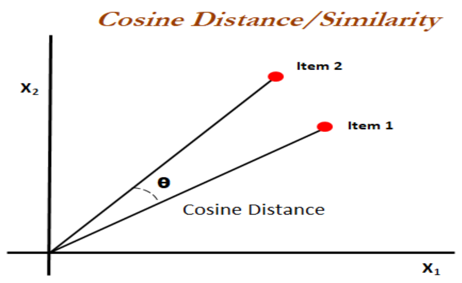
-We were looking for a manner to further group emails related to a specific keyword.

The first thing that came to mind to achieve this was cosine similarity.

A common technique used to measure cohesion within clusters in the field of data mining.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}},$$

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. The cosine of 0° is 1, and it is less than 1 for any other angle.



To find the cosine distance of one email and all the others, We just need to compute the dot products of the first vector with all of the others as the vectors are already row-normalized. To get the first vector, We need to slice the matrix row-wise to get a submatrix with a single row.Using cosine similarity we will find the similarity between two e-mails.

SUPERVISED LEARNING:

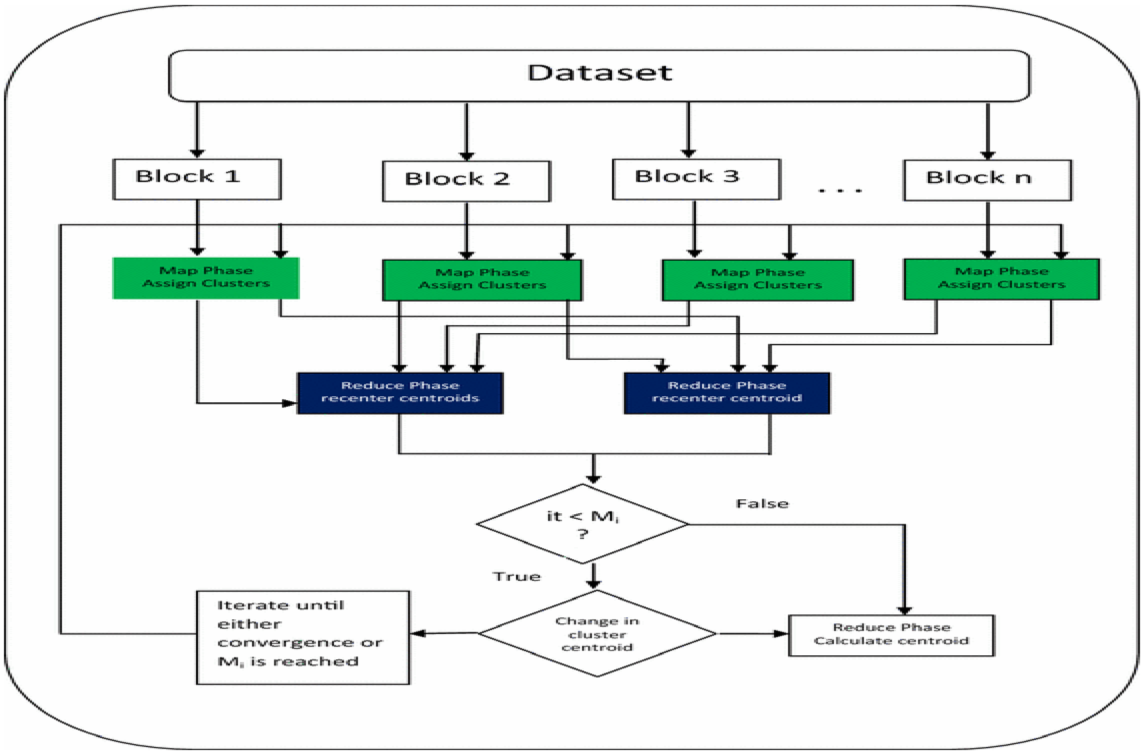
We are going to group the e-mails based on the particular key word.For this we used supervised learning.

In supervised learning a teacher or oracle is available which provides the desired action corresponding to a perception. A set of perception action pair provides what is called a training set.

Examples include an automated vehicle where a set of vision inputs and the corresponding steering actions are available to the learner.

DESIGN :

Usecase diagram

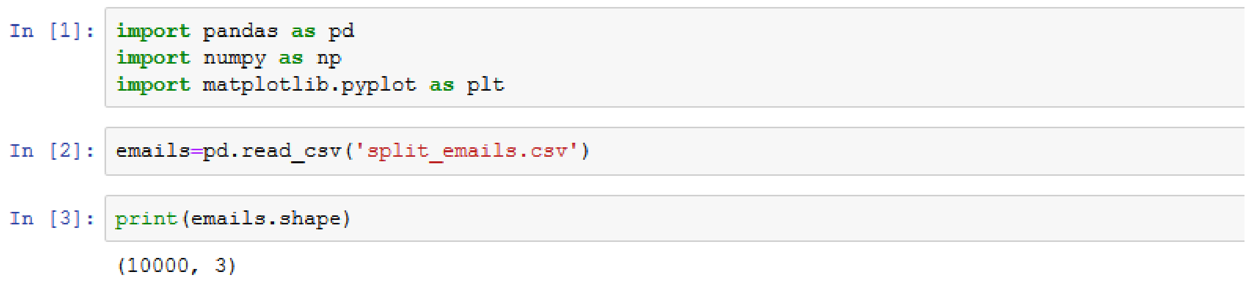

Dataset
Block 1
Block 2
Block 3
. . .
Block n
Map Phase Assign Clusters
Map Phase Assign Clusters
Map Phase Assign Clusters
Map Phase Assign Clusters
Reduce Phase recenter centroids
Reduce Phase recenter centroid
it < M_i ?
False
True
Change in cluster centroid
Iterate until either convergence or M_i is reached
Reduce Phase Calculate centroid

EXPLANATION AND TESTING:

The first thing we did was look for a dataset that contained a good variety of emails. After looking into several datasets, we came up with the Enron corpus. This dataset has over 500,000 emails generated by employees of the Enron Corporation.

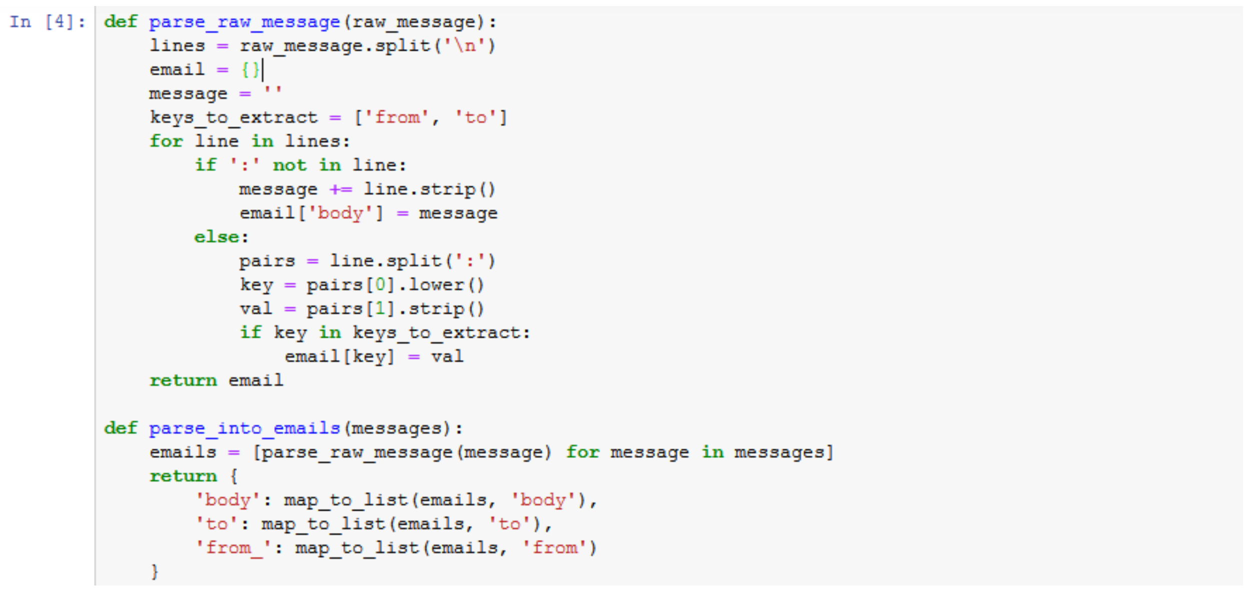
As the programming language, we used Python along with its great libraries: scikit-learn, pandas, numpy and matplotlib.

Instead of loading in all +500k emails, we loaded the dataset into a couple of files with each 10k emails.



```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt

In [2]: emails=pd.read_csv('split_emails.csv')

In [3]: print(emails.shape)

        (10000, 3)
```

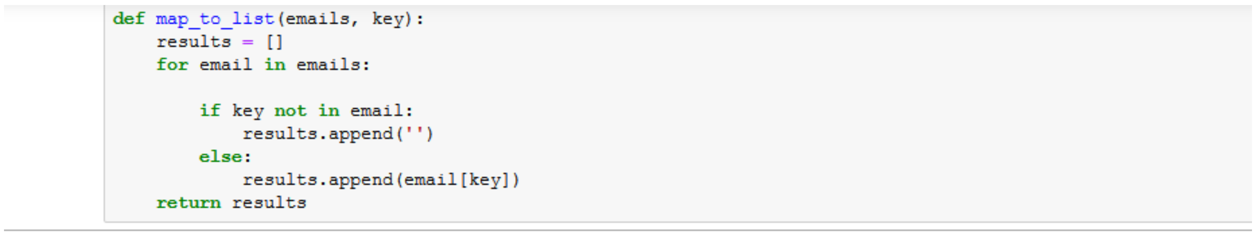
We now had 10k emails in the dataset separated into 3 columns (index, message_id and the raw message). Before working with this data we parsed the raw message into key-value pairs.



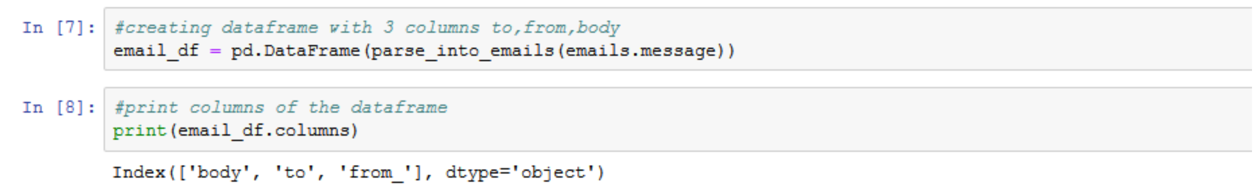
```
In [4]: def parse_raw_message(raw_message):
            lines = raw_message.split('\n')
            email = {}
            message = ''
            keys_to_extract = ['from', 'to']
            for line in lines:
                if ':' not in line:
                    message += line.strip()
                    email['body'] = message
                else:
                    pairs = line.split(':')
                    key = pairs[0].lower()
                    val = pairs[1].strip()
                    if key in keys_to_extract:
                        email[key] = val
            return email

        def parse_into_emails(messages):
            emails = [parse_raw_message(message) for message in messages]
            return {
                'body': map_to_list(emails, 'body'),
                'to': map_to_list(emails, 'to'),
                'from_': map_to_list(emails, 'from')
            }
```

To work with only the sender, receiver and email body data, we made a function that extracts these data into key-value pairs.

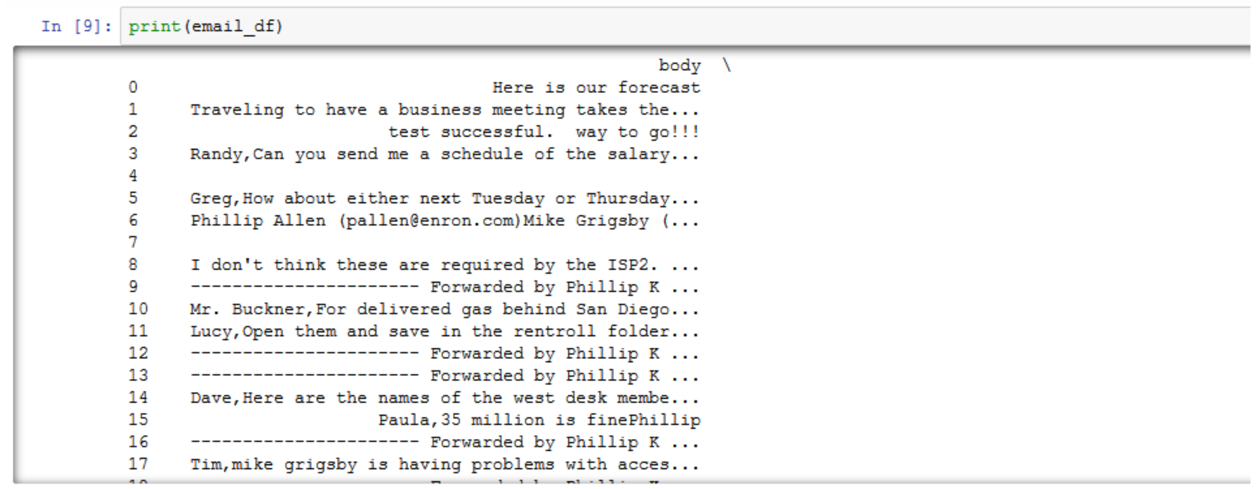
After running this function, we created a new dataframe which is shown below.



```
def map_to_list(emails, key):
    results = []
    for email in emails:

        if key not in email:
            results.append('')
        else:
            results.append(email[key])
    return results
```

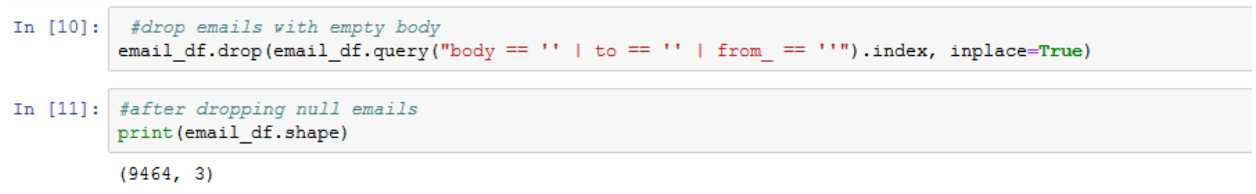


```
In [7]: #creating dataframe with 3 columns to,from,body
        email_df = pd.DataFrame(parse_into_emails(emails.message))

In [8]: #print columns of the dataframe
        print(email_df.columns)

        Index(['body', 'to', 'from_'], dtype='object')
```



```
In [9]: print(email_df)

                                                   body  \
0                              Here is our forecast
1     Traveling to have a business meeting takes the...
2                         test successful.  way to go!!!
3     Randy,Can you send me a schedule of the salary...
4
5     Greg,How about either next Tuesday or Thursday...
6     Phillip Allen (pallen@enron.com)Mike Grigsby (...
7
8     I don't think these are required by the ISP2. ...
9     ---------------------- Forwarded by Phillip K ...
10    Mr. Buckner,For delivered gas behind San Diego...
11    Lucy,Open them and save in the rentroll folder...
12    ---------------------- Forwarded by Phillip K ...
13    ---------------------- Forwarded by Phillip K ...
14    Dave,Here are the names of the west desk membe...
15                     Paula,35 million is finePhillip
16    ---------------------- Forwarded by Phillip K ...
17    Tim,mike grigsby is having problems with acces...
```

To be 100% sure there are no empty columns:



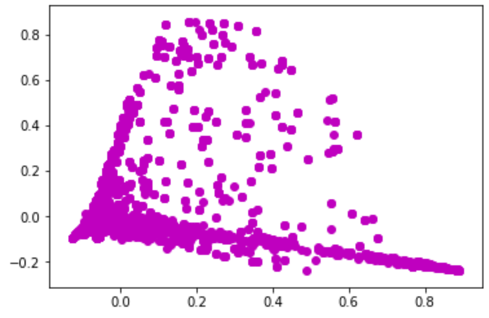
```
In [10]:  #drop emails with empty body
         email_df.drop(email_df.query("body == '' | to == '' | from_ == ''").index, inplace=True)

In [11]: #after dropping null emails
         print(email_df.shape)

         (9464, 3)
```

We need to feed the machine something it can understand, machines are bad with text, but they shine with numbers. Which is why we converted the email bodies into a document-term matrix.

We made a quick plot to visualize this matrix. To do this I first needed to make a 2d representation of the DTM (document-term matrix).

```
In [12]: from sklearn.feature_extraction.text import TfidfVectorizer, ENGLISH_STOP_WORDS
         vect = TfidfVectorizer(stop_words='english', max_df=0.50, min_df=2)
         X = vect.fit_transform(email_df.body)
```

```
In [13]: from sklearn.decomposition import PCA
         X_dense = X.todense()
         coords = PCA(n_components=2).fit_transform(X_dense)

         plt.scatter(coords[:, 0], coords[:, 1], c='m')
         plt.show()
```



To find out what the top keywords were in those emails:

```
In [14]: def top_tfidf_feats(row, features, top_n=20):
             topn_ids = np.argsort(row)[::-1][:top_n]
             top_feats = [(features[i], row[i]) for i in topn_ids]
             df = pd.DataFrame(top_feats, columns=['features', 'score'])
             return df

         def top_feats_in_doc(X, features, row_id, top_n=25):
             row = np.squeeze(X[row_id].toarray())
             return top_tfidf_feats(row, features, top_n)
```

```
In [15]: features = vect.get_feature_names()
         print(top_feats_in_doc(X,features,1,10))
```

```
    features     score
0   meetings  0.383128
1       trip  0.324351
2        ski  0.280451
3   business  0.276205
4      takes  0.204126
5        try  0.161225
6  presenter  0.158455
7  stimulate  0.155878
8      quiet  0.148051
9     speaks  0.148051
```

To get the top terms out of all the emails:

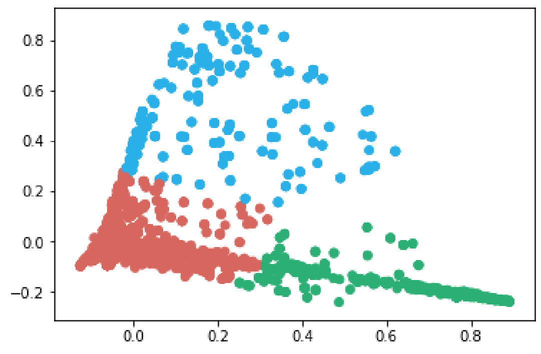
```
In [16]: def top_mean_feats(X, features, grp_ids=None, min_tfidf=0.1, top_n=25):
             if grp_ids:
                 D = X[grp_ids].toarray()
             else:
                 D = X.toarray()

             D[D < min_tfidf] = 0
             tfidf_means = np.mean(D, axis=0)
             return top_tfidf_feats(tfidf_means, features, top_n)
```

```
In [17]: #top terms out of all mails
         print(top_mean_feats(X, features, top_n=10))
```

```
   features     score
0     enron  0.044036
1       com  0.033229
2       ect  0.027058
3       hou  0.017350
4   message  0.016722
5  original  0.014824
6   phillip  0.012118
7     image  0.009894
8       gas  0.009022
9      john  0.008551
```

KMeans is a popular clustering algorithm used in machine learning, where K stands for the number of clusters. I created a KMeans classifier with 3 clusters and 150 iterations.

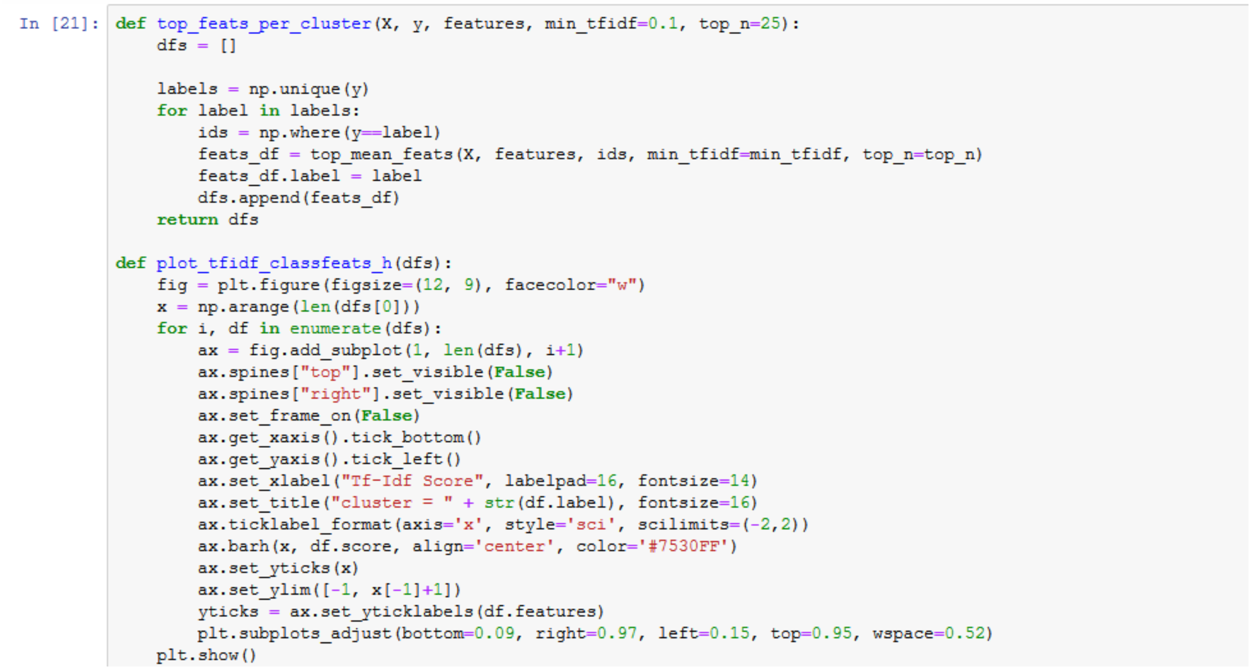
```
In [20]: from sklearn.decomposition import PCA
         centroids = clf.cluster_centers_
         centroid_coords = PCA(n_components=2).fit_transform(centroids)
         plt.scatter(coords[:, 0],coords[:, 1],color=colors)
         plt.show()
```



```
In [18]: from sklearn.cluster import KMeans, MiniBatchKMeans
         n_clusters = 3
         clf = KMeans(n_clusters=n_clusters,
                     max_iter=150,
                     init='k-means++',
                     n_init=1)
         labels = clf.fit_predict(X)
```

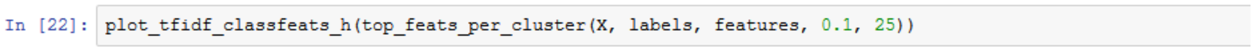
```
In [19]: label_colors = ["#2AB0E9", "#2BAF74", "#D7665E", "#CCCCCC",
                         "#D2CA0D", "#522A64", "#A3DB05", "#FC6514"]
         colors = [label_colors[i] for i in labels]
```

Because we now knew which emails the machine assigned to each cluster, we were able to write a function that extracts the top terms per cluster.

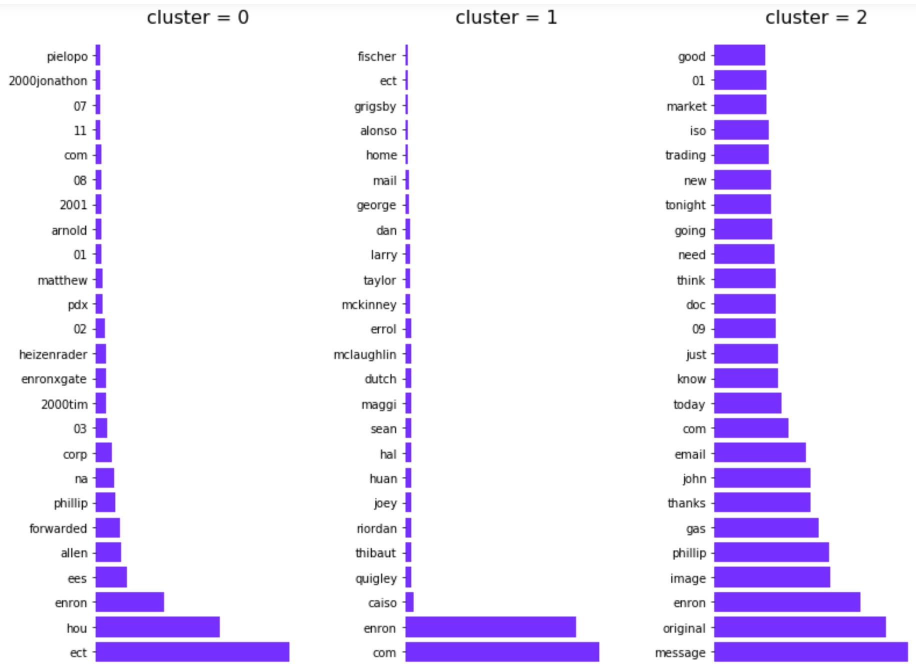


```
In [21]: def top_feats_per_cluster(X, y, features, min_tfidf=0.1, top_n=25):
             dfs = []

             labels = np.unique(y)
             for label in labels:
                 ids = np.where(y==label)
                 feats_df = top_mean_feats(X, features, ids, min_tfidf=min_tfidf, top_n=top_n)
                 feats_df.label = label
                 dfs.append(feats_df)
             return dfs

         def plot_tfidf_classfeats_h(dfs):
             fig = plt.figure(figsize=(12, 9), facecolor="w")
             x = np.arange(len(dfs[0]))
             for i, df in enumerate(dfs):
                 ax = fig.add_subplot(1, len(dfs), i+1)
                 ax.spines["top"].set_visible(False)
                 ax.spines["right"].set_visible(False)
                 ax.set_frame_on(False)
                 ax.get_xaxis().tick_bottom()
                 ax.get_yaxis().tick_left()
                 ax.set_xlabel("Tf-Idf Score", labelpad=16, fontsize=14)
                 ax.set_title("cluster = " + str(df.label), fontsize=16)
                 ax.ticklabel_format(axis='x', style='sci', scilimits=(-2,2))
                 ax.barh(x, df.score, align='center', color='#7530FF')
                 ax.set_yticks(x)
                 ax.set_ylim([-1, x[-1]+1])
                 yticks = ax.set_yticklabels(df.features)
                 plt.subplots_adjust(bottom=0.09, right=0.97, left=0.15, top=0.95, wspace=0.52)
             plt.show()
```

We plot the terms as:

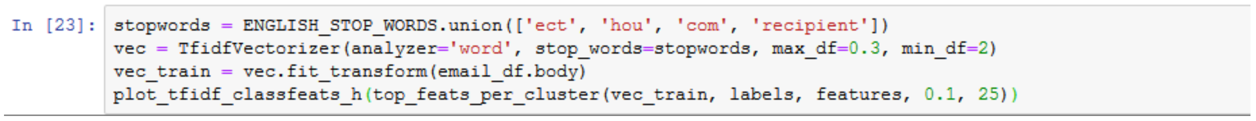


```
In [22]: plot_tfidf_classfeats_h(top_feats_per_cluster(X, labels, features, 0.1, 25))
```



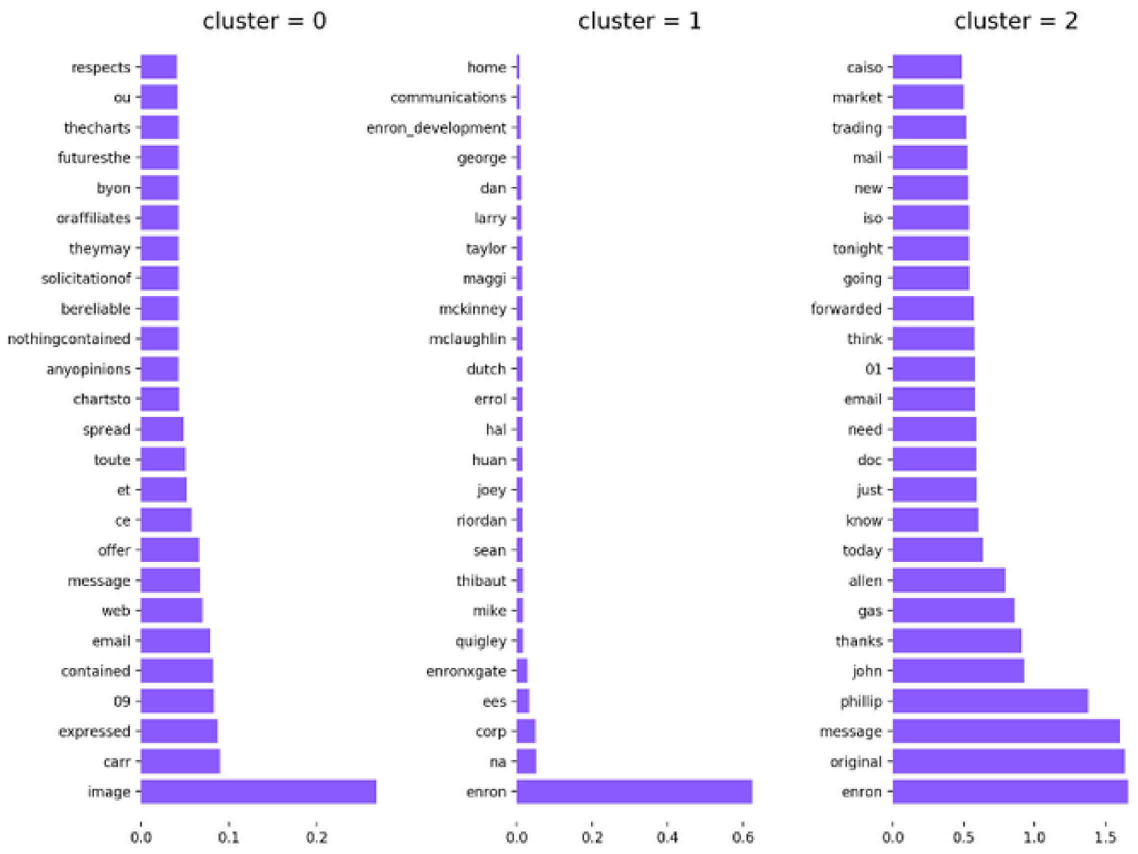
We noticed **cluster 1**, had weird terms like ‘hou’ and ‘ect’. To get more insights about why terms like ‘hou’ and ‘ect’ are so popular, we basically needed to get more insight in the whole dataset, implying a different approach..

After looking through some emails in the dataset, it was apparent why those terms are top hits. They are almost in every TO, CC or BCC rule. To fix this, we added some custom stopwords to the tfidf vectorizer. Because the stopwords are a frozen list, we made a copy of it and passed it to the vectorizer.



```
In [23]: stopwords = ENGLISH_STOP_WORDS.union(['ect', 'hou', 'com', 'recipient'])
         vec = TfidfVectorizer(analyzer='word', stop_words=stopwords, max_df=0.3, min_df=2)
         vec_train = vec.fit_transform(email_df.body)
         plot_tfidf_classfeats_h(top_feats_per_cluster(vec_train, labels, features, 0.1, 25))
```

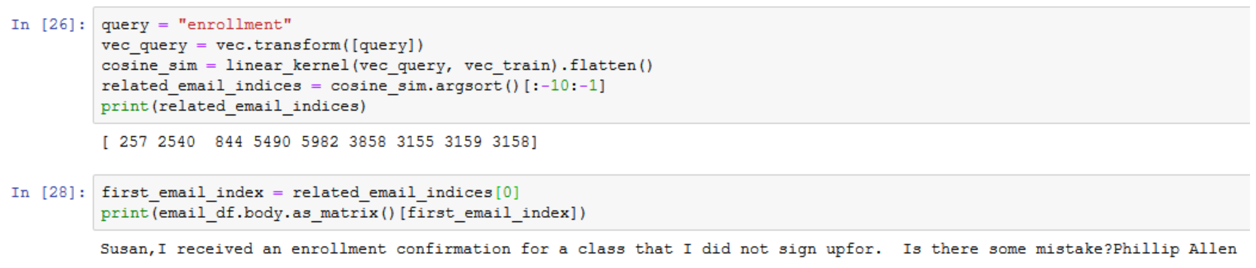
After this we will get updated clusters as follows:



Now we had some insights on how those emails where clustered, it was time to take my research a little step further. After discovering the most popular terms and the most exciting emails due to clustering algorithms, we were looking for a manner to further group emails related to a specific keyword.

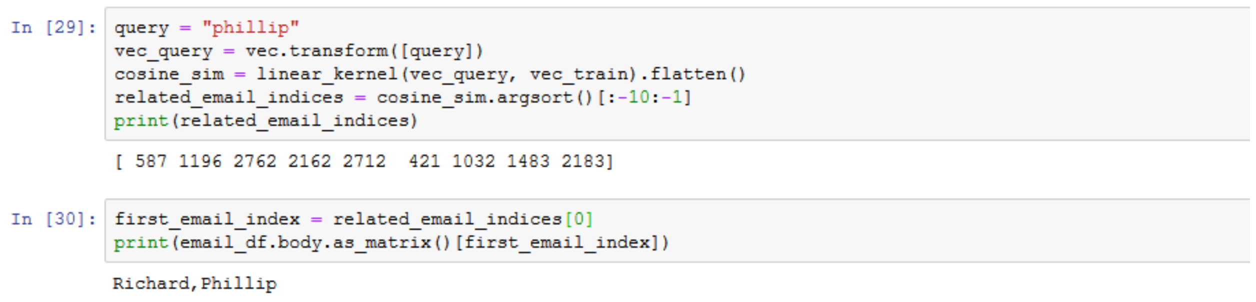
Instead of finding emails related to each other, we want to see emails pertaining to a “query” (e.g., a specific keyword or term) that we can specify.

To find the top 10 emails that match our query we used argsort and some negative array slicing (most relevant emails have higher cosine similarity values).Lets take some queries and see the outputs.

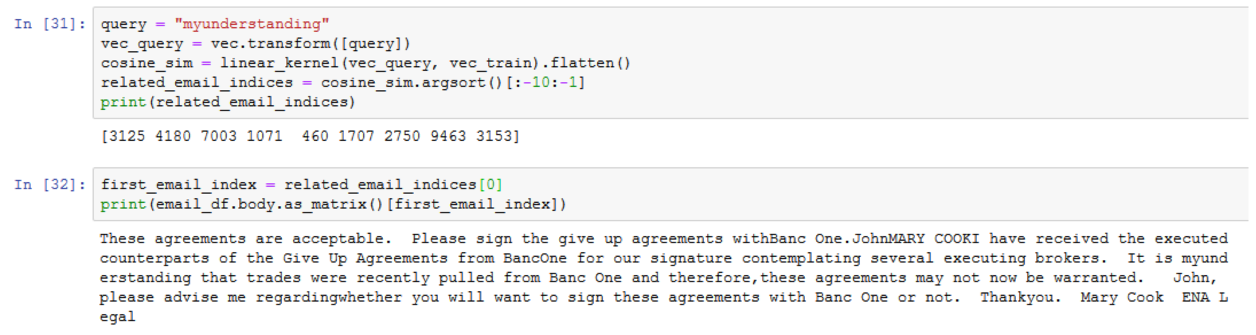


```
In [26]: query = "enrollment"
         vec_query = vec.transform([query])
         cosine_sim = linear_kernel(vec_query, vec_train).flatten()
         related_email_indices = cosine_sim.argsort()[:-10:-1]
         print(related_email_indices)

         [ 257 2540  844 5490 5982 3858 3155 3159 3158]

In [28]: first_email_index = related_email_indices[0]
         print(email_df.body.as_matrix()[first_email_index])

         Susan,I received an enrollment confirmation for a class that I did not sign upfor.  Is there some mistake?Phillip Allen
```

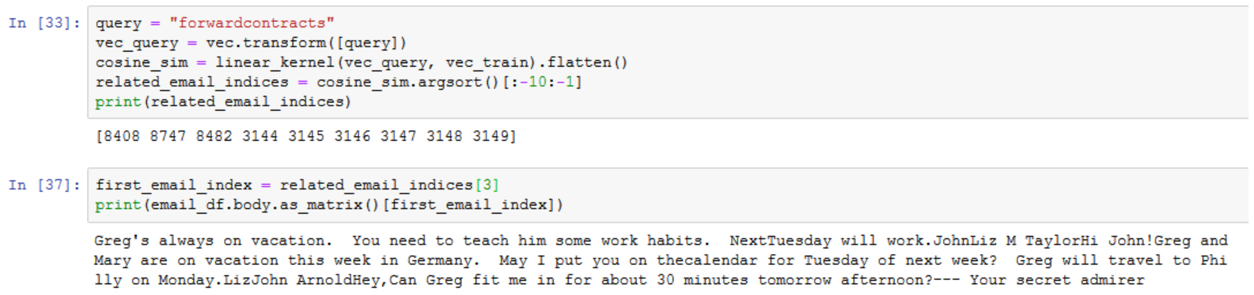
TESTCASES AND OUTPUTS:



```
In [29]: query = "phillip"
         vec_query = vec.transform([query])
         cosine_sim = linear_kernel(vec_query, vec_train).flatten()
         related_email_indices = cosine_sim.argsort()[:-10:-1]
         print(related_email_indices)

         [ 587 1196 2762 2162 2712  421 1032 1483 2183]

In [30]: first_email_index = related_email_indices[0]
         print(email_df.body.as_matrix()[first_email_index])

         Richard,Phillip
```



```
In [31]: query = "myunderstanding"
         vec_query = vec.transform([query])
         cosine_sim = linear_kernel(vec_query, vec_train).flatten()
         related_email_indices = cosine_sim.argsort()[:-10:-1]
         print(related_email_indices)

         [3125 4180 7003 1071  460 1707 2750 9463 3153]

In [32]: first_email_index = related_email_indices[0]
         print(email_df.body.as_matrix()[first_email_index])

         These agreements are acceptable.  Please sign the give up agreements withBanc One.JohnMARY COOKI have received the executed
         counterparts of the Give Up Agreements from BancOne for our signature contemplating several executing brokers.  It is myund
         erstanding that trades were recently pulled from Banc One and therefore,these agreements may not now be warranted.   John,
         please advise me regardingwhether you will want to sign these agreements with Banc One or not.  Thankyou.  Mary Cook  ENA L
         egal
```



```
In [33]: query = "forwardcontracts"
         vec_query = vec.transform([query])
         cosine_sim = linear_kernel(vec_query, vec_train).flatten()
         related_email_indices = cosine_sim.argsort()[:-10:-1]
         print(related_email_indices)

         [8408 8747 8482 3144 3145 3146 3147 3148 3149]

In [37]: first_email_index = related_email_indices[3]
         print(email_df.body.as_matrix()[first_email_index])

         Greg's always on vacation.  You need to teach him some work habits.  NextTuesday will work.JohnLiz M TaylorHi John!Greg and
         Mary are on vacation this week in Germany.  May I put you on thecalendar for Tuesday of next week?  Greg will travel to Phi
         lly on Monday.LizJohn ArnoldHey,Can Greg fit me in for about 30 minutes tomorrow afternoon?--- Your secret admirer
```

CONCLUSION:

In order to compare new techniques for email classification, a large standard test dataset, such as the newly available Enron corpus, could be very valuable.

It can be used both as a large sample of real life email users and as a standard corpus for comparison of results using different methods. We have provided evaluations of baseline experiments on the dataset. We have also provided a brief analysis of the dataset itself.